

# Face tracking

(In the context of Saya, the android secretary)

Anton Podolsky and Valery Frolov

## ❖ Introduction

Given the rather ambitious task of developing a robust face tracking algorithm which could be seamlessly integrated with Saya's vision system, we embarked on a journey of exploration of the various techniques kindly available for research. In light of the fact that first attempts at face detection date back to the late 90's, the amount of data archived so far was quite overwhelming.

Our first attempts included subtraction of subsequent frames and subtraction of frames against a predefined background. These ideas failed to achieve the basic goals and were quickly banned.

The next attempt was to implement a method based on skin color filtering in chromatic color space and Gaussian skin color model[1][2]. This method was chosen mostly for its ability to be unaffected by luminosity changes. While looking promising at first and given a great deal of attention and effort, it was also rejected in favor of a method based on Haar-like features[3] which will be discussed in this report.

## ❖ Motivation

Saya is a high-tech I/O device which aspires to imitate human behavior. We try to instill a specific human-like property of eyes and head movement in reaction to human presence. The main challenge therefore is the detection of human faces.

## ❖ Technique overview

As proposed by P. Viola in 2001, we are using Haar-like features based technique to describe human faces.

- Definitions:

- AdaBoost – Learning algorithm for boosting the classification performance of a simple learning algorithm. It does this by combining a collection of weak classification functions to form a stronger classifier.
- Haar-like features - Digital image features used in object recognition. A set of Haar-like features considers rectangular regions of the image and sums up the pixels in this region. The sum is used to categorize images. In a sample 20x20 image there are over 40,000 features, from which we have to choose up to 200 “good” features.
- Classifier cascade - A cascade of classifiers is a degenerated decision tree where at each stage a classifier is trained to detect almost all objects of interest (frontal faces in our example) while rejecting a certain fraction of the non-object patterns.

- The algorithm:

The goal is to construct a classifier cascade with overall high detection rate and low false-positive rate, while each level in the cascade holds a strong classifier consisting of groups of weak classifiers. Each weak classifier is bounded to a single Haar-like feature. AdaBoost procedure is used to choose “better” features to form a stronger classifier.

- Training the cascade:

Two sets of grayscale sample images are used in the training process. One of human face samples and the other of arbitrary non-face samples.

At level  $i$ , AdaBoost is used to build a strong classifier. The classifier is tested on the same samples. Non-face samples which were classified incorrectly are used to create a set of non-face images for stage  $i + 1$ . Face samples remain the same.

To achieve an overall 0.9 detect rate for a 10 level cascade, each level has to detect at least 99% of faces from the training set. On the other hand, to achieve 1 to 1,000,000 false-positive rate, each level is allowed to false-alarm only about 30% of the non-face samples.

- Building a strong classifier:

Given a training set of weighted positive and negative examples and a set of weak classifiers, at iteration  $i$  AdaBoost chooses the classifier with best performance on the training set among the set of weak classifiers. Weights of incorrectly classified samples are increased, so at next iteration we focus the attention on “harder” examples. Final strong classifier consists of the best weak classifiers chosen at each iteration.

- Training an individual classifier:

Each classifier is a simple perceptron. The training process tries to find an optimal threshold value which minimizes the error rate over the training set.

## ❖ Architecture

Two approaches were taken here.

One has taken the path of implementing the described above technique from scratch in Java but due to time/resources constraints we couldn't deliver a finished product.

The other approach heavily relies on the OpenCV C library by Intel, which implements the same basic idea.

### **The latter:**

The original OpenCV code responsible for face detection was slightly changed by introducing some logic for integration with Saya, compiled into a dll which was encapsulated by Java using the JNI library. The Java wrapper is basically responsible for interaction with Saya's API, i.e. sending the desired angle (though it lacks the actual message sending code).

The changes are as follows:

1. Face tracking algorithm:
  - a. Wait for a face(s) to appear in the frame
  - b. Enter initialization mode (wait for a face(s) to appear for a predefined amount of time, to avoid paying attention to people who just happen to pass by)
  - b. Enter tracking mode, choose the closest face.
  - c. Track the face until it leaves the frame. To avoid losing track of the face due to minor head movements (e.g. briefly looking sideways), leave tracking mode only when the tracked face disappears for a predefined amount of time.
  - d. Go to a.
2. Angle calculation.

### **The former:**

Description of the main classes:

**CascadeTrainer.java** – implements a P.Viola and M.Jones algorithm [3] for training a cascade of classifiers

**AdaBoost.java** – a modified version of the original AdaBoost algorithm. Used to select the Haar-like features and to train the classifier.

**IntegralImage.java** – implements a P.Viola intermediate representation of an image [3] for rapid rectangular features calculation.

**Feature.java** – Haar-like features generator for any given size, e.g. 19x19.

**Scanner.java** – returns a rectangular representation of each face found in an input image by shifting a detection window of various sizes through the image.

The rest of the classes provide helper functions and data structures.

The training set consists of 4,000 face and 4,000 non-face 19x19 grayscale samples, downloaded from some good folks on the net.

## ❖ Results

### **OpenCV based, allegedly working program:**

The performance of the OpenCV library is, as expected, fast and accurate with a 320x240 webcam on a low-end PC. Therefore, its integration with Saya shouldn't pose any problem.

All tests were based on imitating (as close as possible) head movement and of course the final behavior of the algorithm may not be as precise as intended.

### **Java implementation:**

As mentioned earlier this isn't a complete working product but rather a solid fundament for future development.

#### *What we achieved so far:*

Full implementation of P.Viola and M.Jones method. Including cascade training, a variant of AdaBoost and a multi-scale scanner.

#### *What needs to be completed:*

An actual high performance (at least 90% detection and at most 1 to 1,000,000 false-positive rates) trained cascade and further testing and profiling. Integration with Saya's camera through the JMF API. Java implementation of the previously described face tracking algorithm.

## ❖ Conclusion

The cascade trainer showed tolerable performance while training cascade with low-detection and high false-positive rates. Training high performance cascade running times were unbearable. Therefore, scanner consisting of low performance cascade yielded high error rates on 320x240 webcam images.

Retrospectively speaking, there are two main approaches we could have looked into:

1. Implementing the Viola algorithm in a native language for better control over the resources and more efficient calculations.
2. Skipping the process of training our own cascade and writing a loader for one of the high performance cascades already provided by the OpenCV project. This approach would've allowed us to implement the actual detection algorithm in Java for easier integration with Saya main program.

## ❖ Bibliography

- [1] "Face detection in complex background based on Gaussian models and neural networks" (Qi Li, Hongbing Ji)
- [2] "Face tracking in color video sequence" (Peter Gejgus, Martin Sperka)
- [3] "Rapid object detection using boosted cascade of simple features"(Paul Viola, Michael Jones)

## ❖ Sources

- OpenCV Library - <http://en.wikipedia.org/wiki/OpenCV>
- JNI Library - [http://en.wikipedia.org/wiki/Java\\_Native\\_Interface](http://en.wikipedia.org/wiki/Java_Native_Interface),  
<http://java.sun.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html>